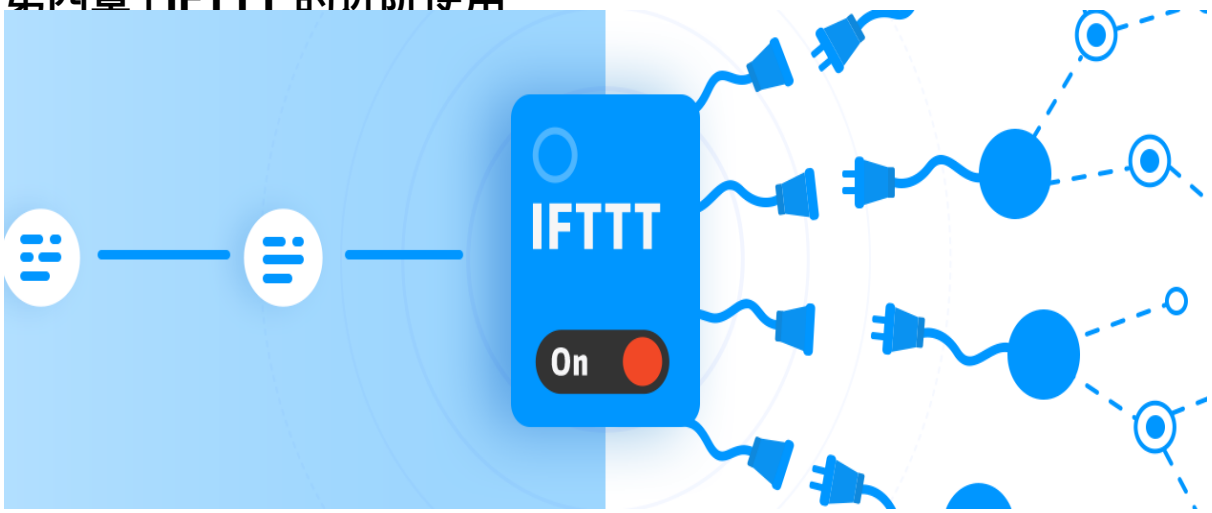
第四章 | IFTTT 的进阶使用



王掌柜

04月18日

第四章 | IFTTT 的进阶使用

| 本文为付费栏目文章，您已订阅，可阅读全文 |

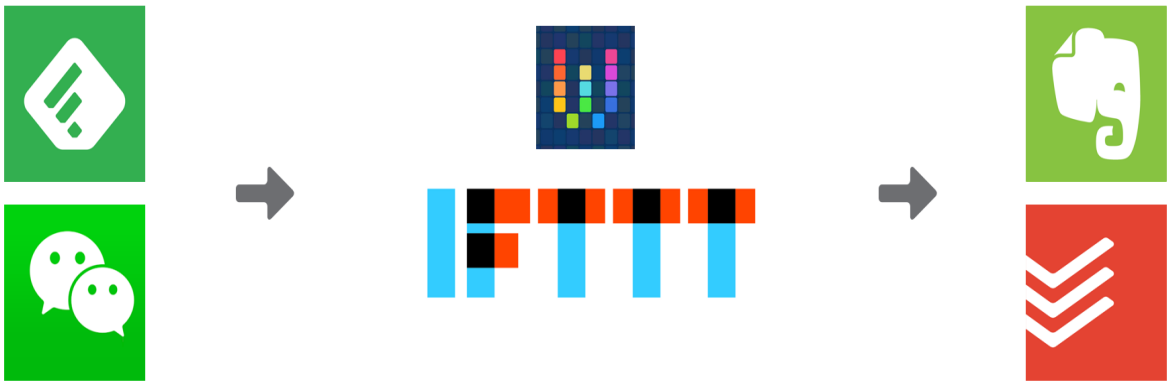
进阶篇的内容会比较复杂，本篇一共有两个事例：

1. 通过串联**多个自动化任务**实现「稍后阅读自动化流程」；
2. IFTTT 的 **Webhooks** 结合 Workflow（iOS 效率神器） 实现「书籍管理自动化」。

在这两个事例里，我们将研究如何串联多个自动化任务实现一套复杂的流程，另外引入 Webhooks 概念，结合 Workflow 实现更加复杂的「本地触发」任务。

事例 1：IFTTT 实现稍后阅读自动化流程

事例 1：IFTTT 实现稍后阅读自动化流程



本例中，我们通过 IFTTT 串联 Feedly[1]、Evernote、Todoist[2] 等服务，实现「稍后阅读自动化流程」。

我的稍后阅读需求

1. 我的碎片化阅读场景包括：
    - 场景 1：Feedly，订阅了许多 RSS 源，定时更新查看。
    - 场景 2：微信公众号，订阅了一些不错的作者，定时更新查看。
    - 场景 3：来自其他途径的阅读，比如来自 Slack、Telgram、微信公众号、微信群、朋友圈的文章等（特指 iOS 使用场景）。
2. 如果当下没有时间阅读，我会将需要稍后处理的文章保存到 Evernote 的「08@稍后读」文件夹，同时设置文章的标签为「稍后读」。
3. 当有一大段充足时间时，我会打开 Evernote 的「08@稍后读」文件夹，找一篇感兴趣的文章进行详细阅读，并且会做笔记。

但是这个流程有一个问题：**待读的文章越攒越多，不能快速消化掉，会带来心理压力，容易引发拖延症。**

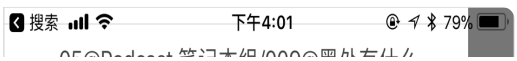
为了解决拖延症，这里引入了「任务管理」的概念，由于我习惯在 Todoist 中

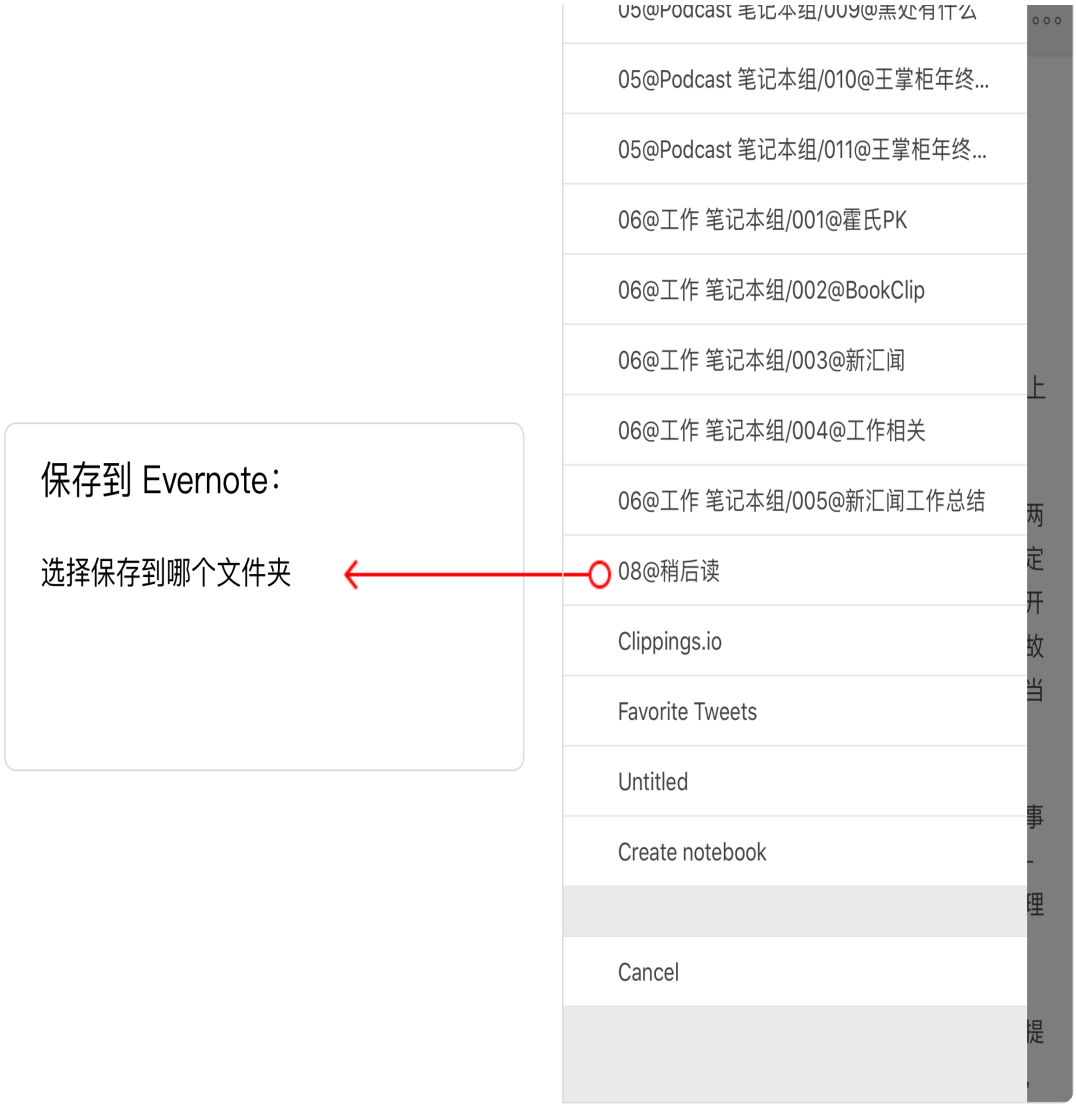
管理所有的任务，所以我会在将文章加入到 Evernote 后，再去 Todoist 中创建一条任务，提醒我当天晚上 22 点阅读文章。这样的话，我每天只需要像做任务一样，打开任务列表找到一篇文章去阅读，读完以后回到任务列表，完成任务即可，非常有成就感，一定程度上能缓解拖延症。

在这个流程里的所有操作我们都可以手动实现：

1. 场景 1：Feedly 保存文章到 Evernote。

菜单里选择保存到 Evernote





选择保存到哪个笔记

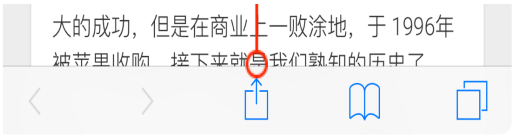
2. 场景 2：将文章从应用保存到 Evernote（以微信为例）。

微信文章保存到 Evernote

点击弹出选项卡里选择「在 Safari 中打开」

Share sheet:

点击弹出 Share sheet 选项卡

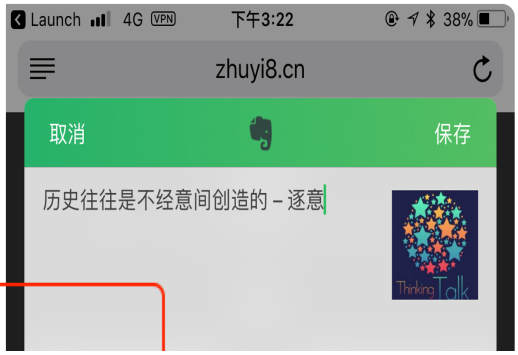


Share sheet:

选择分享到「印象笔记」

Share sheet:

1. 设置保存到哪个文件夹



2. 设置文章的标签

3. 去 Todoist 中创建提醒任务。

创建任务：

填写必要信息创建稍后阅读提醒任务

阮一峰博客文章
https://www.evernote.com/shard/s363/nl/150276631/bf9e8c5e-d297-49a7-844c-22a208a91b2b/

我们看到，每个操作都可以手动，但是实际操作起来非常麻烦，我们能否用已经掌握的「知识」自动化实现这个步骤呢？接下来大家跟着我一步步地实现「稍后阅读自动化」。

场景 1：Feedly 文章保存到 Evernote

刚才我们已经看到 Feedly 如何保存文章到 Evernote，这个操作总共需要三步：

1. 打开菜单。
2. 点击 Evernote 图标。
3. 在弹出对话框中选择「08@稍后读」文件夹。

我们改用 IFTTT 实现，看能否简化这个步骤，先看一下需求：

当在 Feedly 中收藏一篇文章的时候，自动将文章保存到 Evernote 的「08@稍后读」文件夹中，并设置文章的标签为「稍后读」。

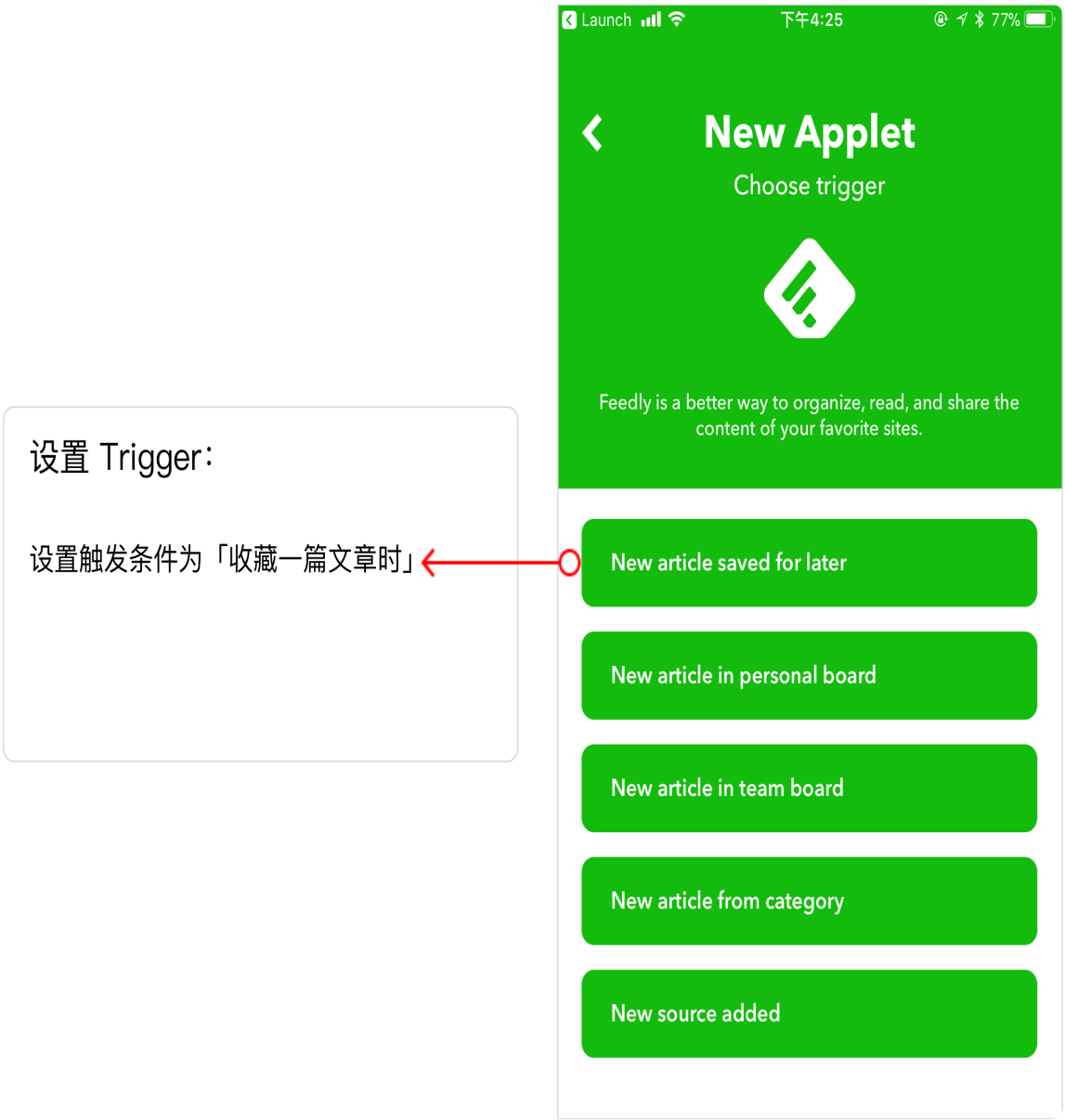
我们分析一下这个需求：

- Trigger：在 Feedly 中收藏一篇文章。
- Action：将收藏的文章保存到 Evernote 的「08@稍后读」中，同时设置

- Action：将收藏的文章保存到 Evernote 的「08@稍后读」中，同时设置标签为的「稍后读」。

设置 Trigger （触发器）[3]

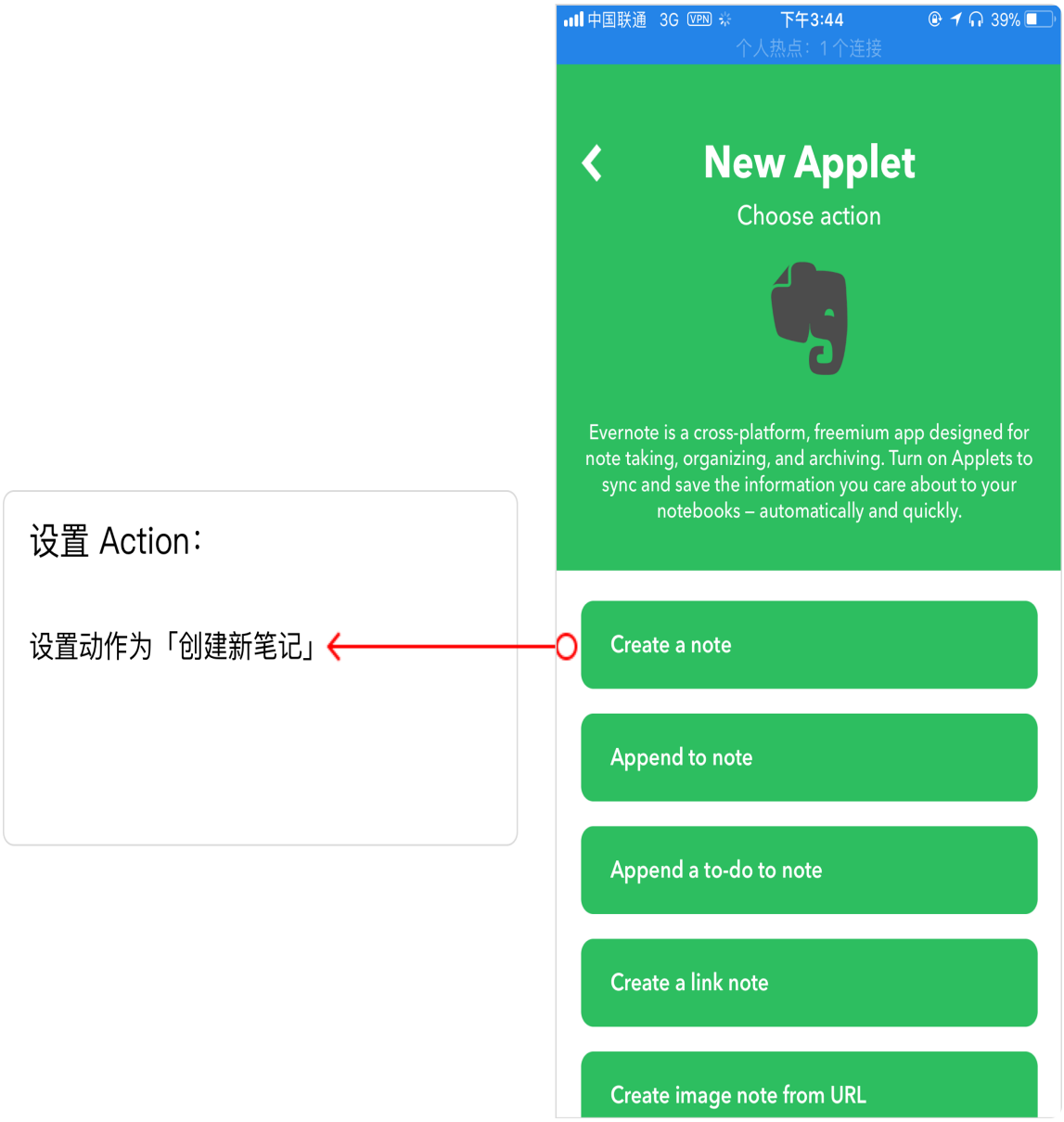
IFTTT 提供了 **Feedly** 服务，我们的触发条件是「当收藏一篇文章时」，Trigger 设置如图：



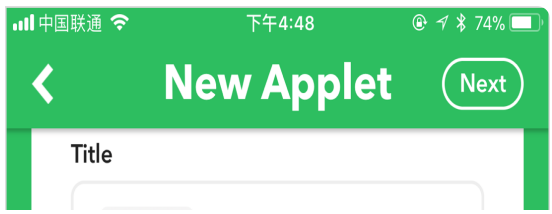
设置 Trigger 为「当收藏一篇文章时」

设置 Action（动作）[4]

我们的动作是将收藏的文章保存到 Evernote 的「08@稍后读」文件夹中，并且设置标签为「稍后读」，IFTTT 已经将绝大多数默认内容给了出来，我们只需要设置文件夹和标签即可，Action 设置如图：



设置 Action 为 「创建一篇新笔记」



设置将哪些内容写进笔记

Action 模版变量说明	
字段	**说明**
ArticleTitle	收藏的 Feedly 文章标题
ArticleURL	收藏的 Feedly 文章地址
ArticleContent	收藏的 Feedly 文章正文
ArticlePersonalBoards	收藏的 Feedly 文章标签

ArticleCategories	收藏的 Feedly 文章类型
SourceTitle	收藏的 Feedly 文章原标题
SourceURL	收藏的 Feedly 文章地址
ArticlePublishedAt	收藏的 Feedly 文章发布时间

场景 1 小结

这一步我们通过 IFTTT 实现了，当在 Feedly 中收藏一篇文章的时候，自动将文章保存到 Evernote 的「08@稍后读」文件夹中，并设置文章的标签为「稍后读」。以后在 Feedly 阅读文章的时候，只需要点击菜单中的「收藏」按钮即可。

通过 IFTTT 我们将原来的三步操作简化为两步，还排除了分享菜单对我们的干扰（无需从众多按钮中找到 Evernote），并且还额外支持设置文章的标签。

场景 2：应用中（以微信为例）的文章保存到 Evernote

刚才我们看到将微信中的文章分享到 Evernote 一共分为 5 步：

1. 在文章页面点击微信右上角「···」按钮。
2. 在弹出页面里选择 Safari 打开。
3. 在浏览器中点击分享按钮（Share Sheet[5]）。
4. 在弹出页面里点击 Evernote 图标。
5. 在弹出层里选择分享到「08@稍后读」以及设置「稍后读」标签。

完成这些操作非常繁琐，但即使这样，这个方法依然还有问题，它不适用分享

「微信公众号」里的文章，由于公众号里的文章格式比较特殊，会导致部分图片无法在 Evernote 中正常显示。

对于这个问题有两种解决方案:

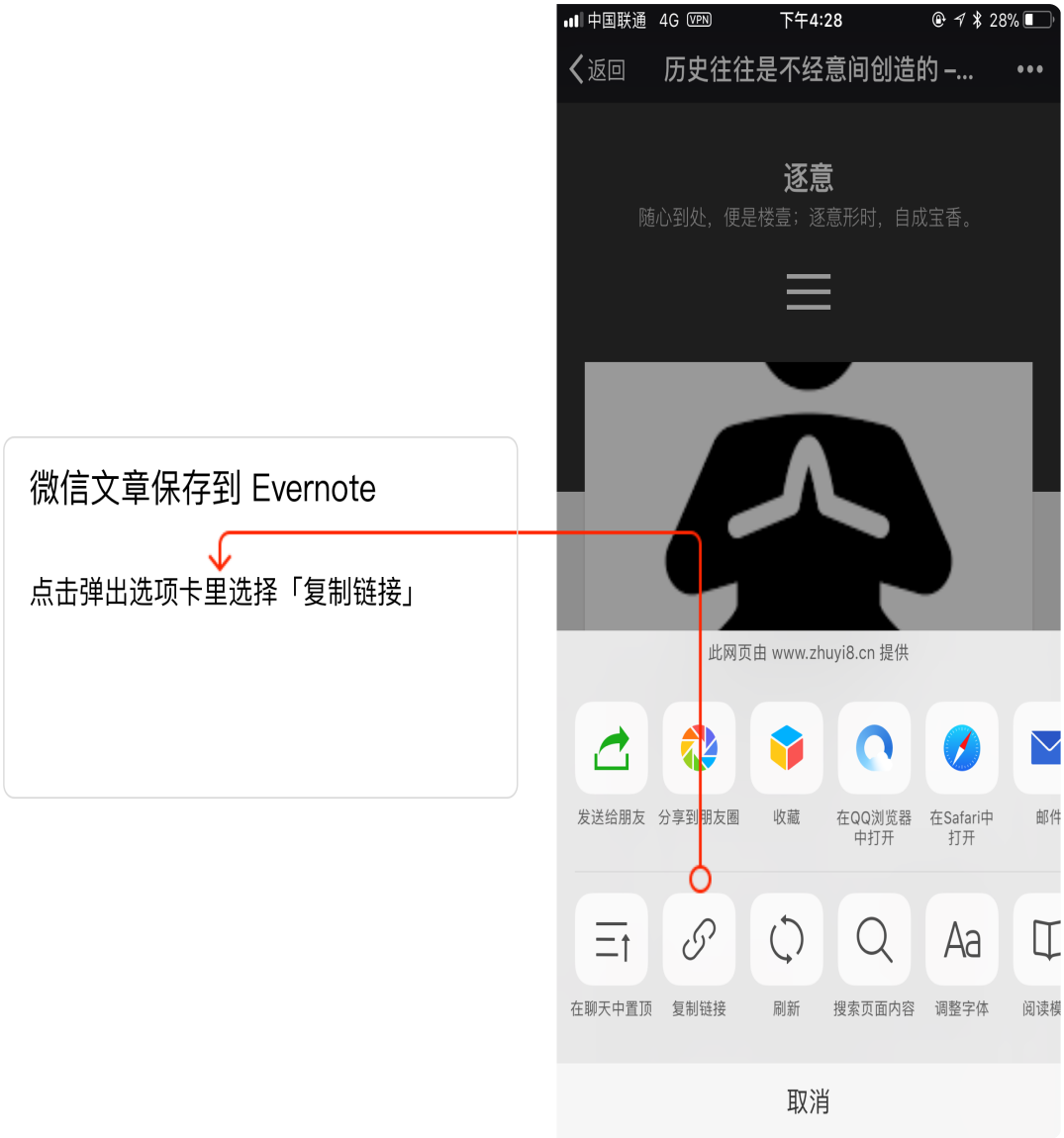
1. 如果你是印象笔记（Evernote 中国版）用户，可以关注「印象笔记」公众号，就会在微信的分享菜单里看到「我的印象笔记」按钮，点击后可一键将文章分享到印象笔记。但是这种方式的弊端是，文章会自动保存到一个叫「微信」的文件夹中，无法自定义。

2. 通过 Workflow 将文章内容抓取解析再上传到 Evernote。它的优点是同时

2. 通过 Workflow 将文章内容抓取解析并上传到 Evernote，它的优点是同时支持「印象笔记」和 「Evernote 国际版」，而且可以自定义保存的文件夹和标签。

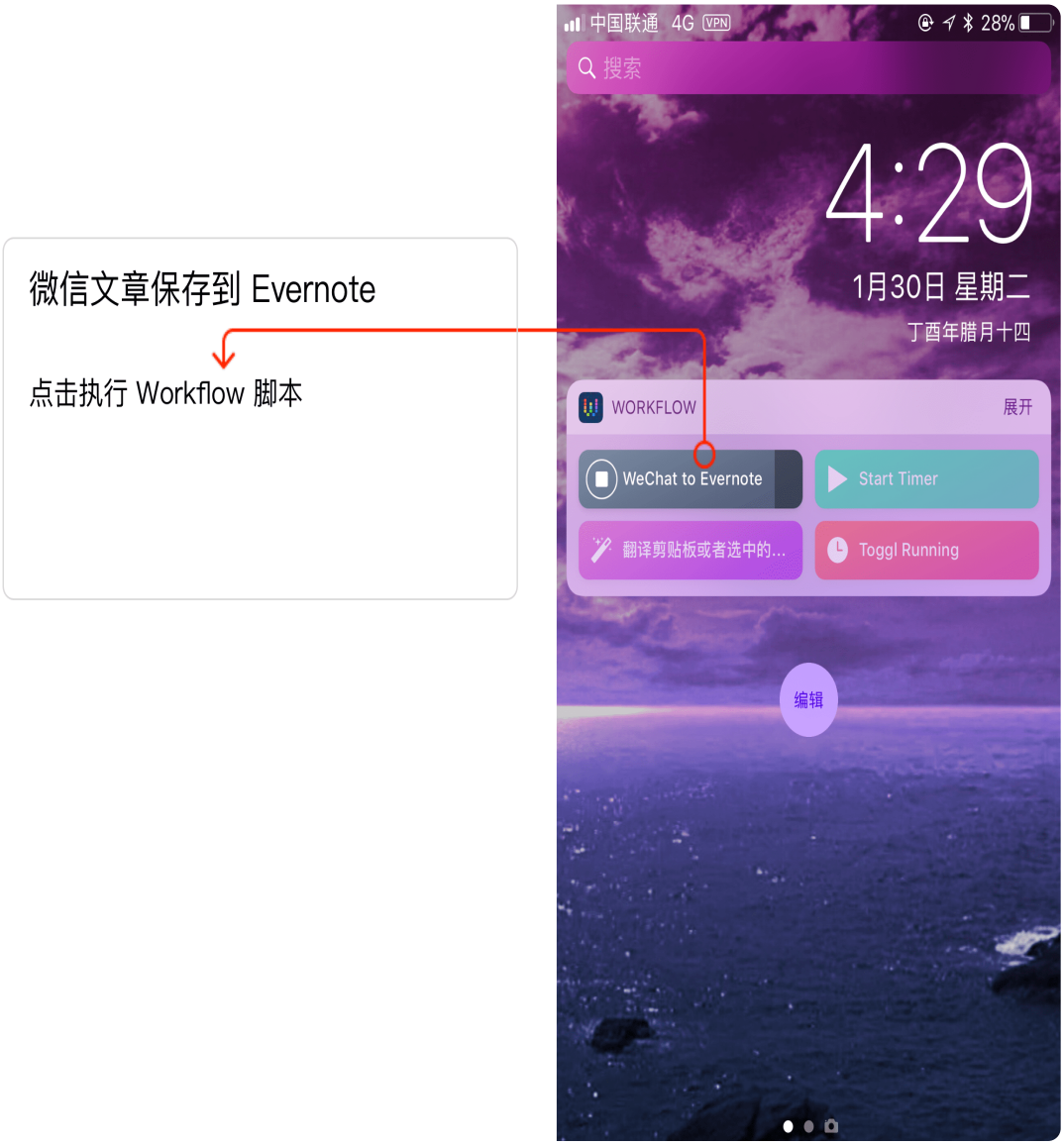
很显然第二种方法更加通用，接下来我们看一下通过 Workflow 分享文章到 Evernote 的步骤：

1. 微信中弹出菜单里选择「复制链接」。



2. 点击 Workflow 的 Widget 中的脚本 **WeChat to Event**，自动将文章保存

到 Evernote「08@稍后读」文件夹中，并设置文章的标签为「稍后读」。



可以看到只需要 2 步即可完成分享，非常高效。

☞ **下载 Workflow 脚本**（感谢作者 @t_bryan）

场景 2 小结

本小节中，我们利用 Workflow，提取文章内容，解析后上传到 Evernote，这种方式支持所有提供了 Share Sheet 的应用。而且完美解决了「微信公众号」

的格式限制，可以算是非常划算地实现「稍后读」的方式了。

文章保存到 Evernote 后创建 Todoist 提醒任务

在上一个自动化流程中，我们已经实现了在 Feedly 中收藏一篇文章或者执行 Workflow 任务，将文章自动保存到 Evernote 中。现在我希望 IFTTT 能自动到 Todoist 中创建一条提醒任务，提醒我当天晚上 10 点阅读 Evernote 中的文章。同时我希望点击任务标题能够跳转到文章的阅读页面。

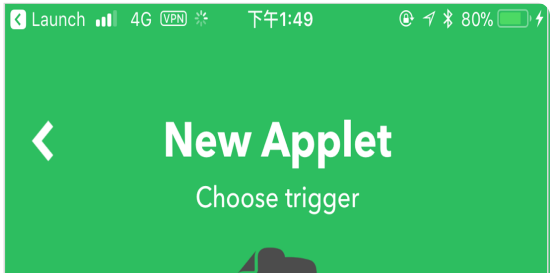
那么问题来了，根据我们的需求，我们可能用到 Feedly、Evernote、和 Todoist 三个服务，但是 IFTTT 只支持两个服务之间的自动化。怎么办呢？我们不妨换一个思路：

以场景1「Feedly 文章保存到 Evernote」为例，我们已经将文章保存到 Evernote 中了，而且设置了文章的标签为「稍后读」，所以第三步我们的需求就可以变成「当 Evernote 中新增了一篇文章且设置文章标签为**稍后读**时，自动去 Todoist 中设置一条提醒任务」，这样 IFTTT 就能完美支持了，我们分析一下这个任务。

- Trigger：当 Evernote 中一篇文章打上「稍后读」标签时。
- Action：在 Todoist 中创建一条提醒任务，提醒我当天晚上 10 点阅读，同时设置任务标题为 Evernote 那篇文章的 URL（可以实现点击标题直接跳转到文章页面）。

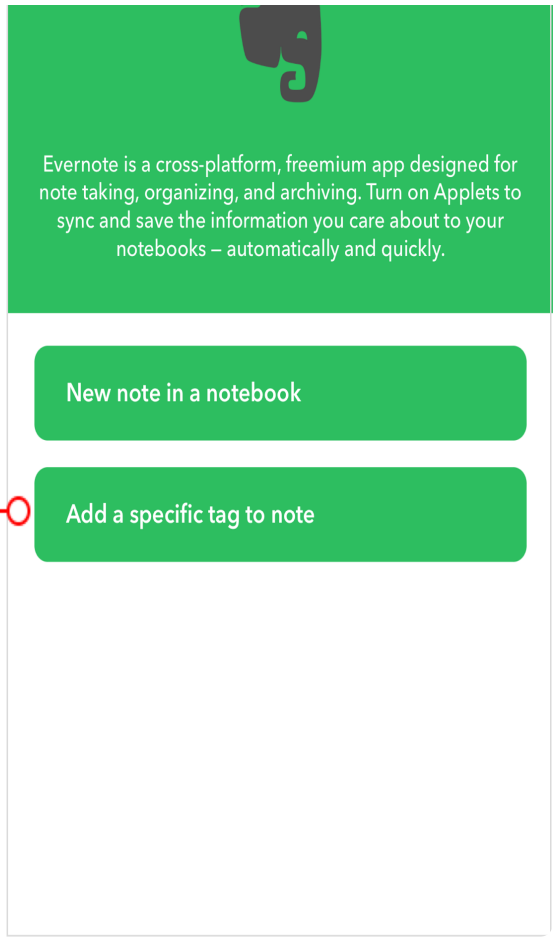
设置 Trigger

设置触发器，选择服务为 Evernote，设置条件是添加新的标签「稍后读」



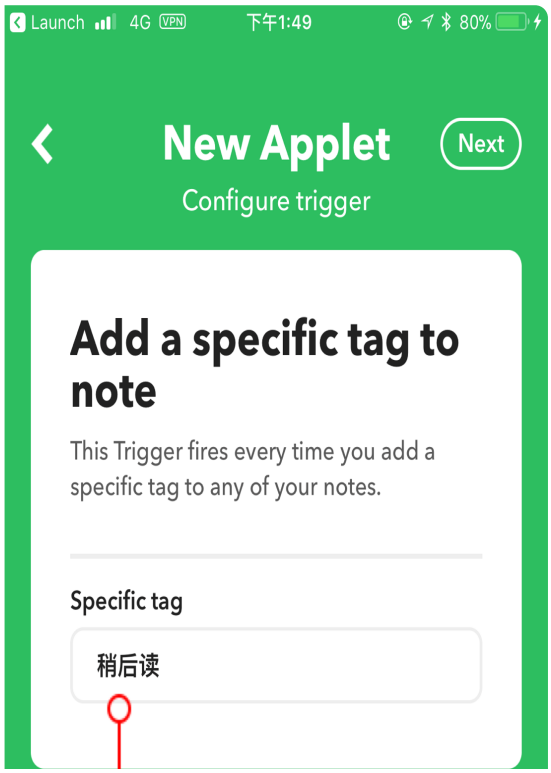
设置 Trigger：

触发条件为「当有笔记设置了指定标签时」



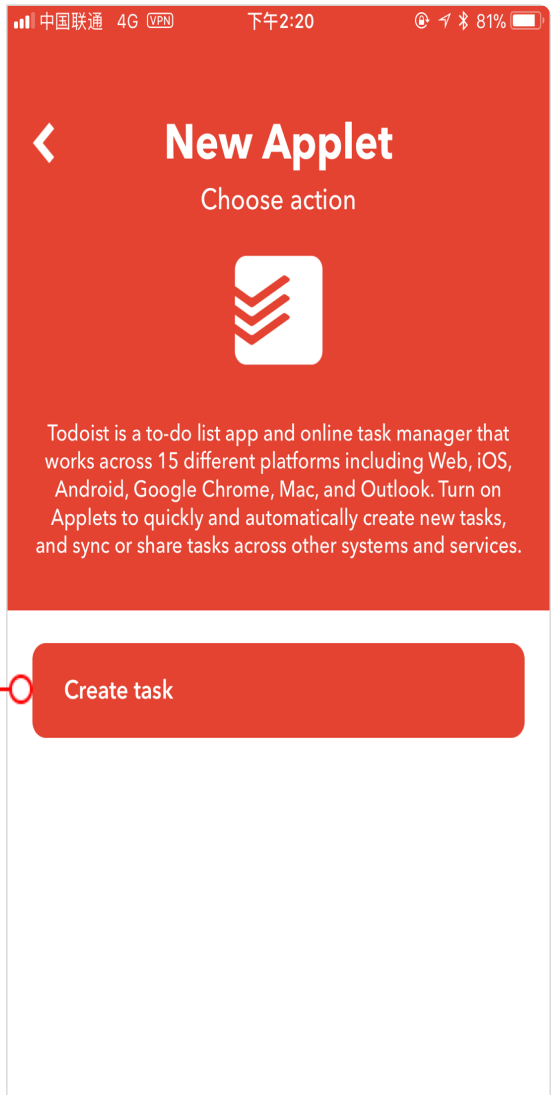
设置 Trigger：

输入需要监控笔记添加了什么「标签」



设置 Action（动作）

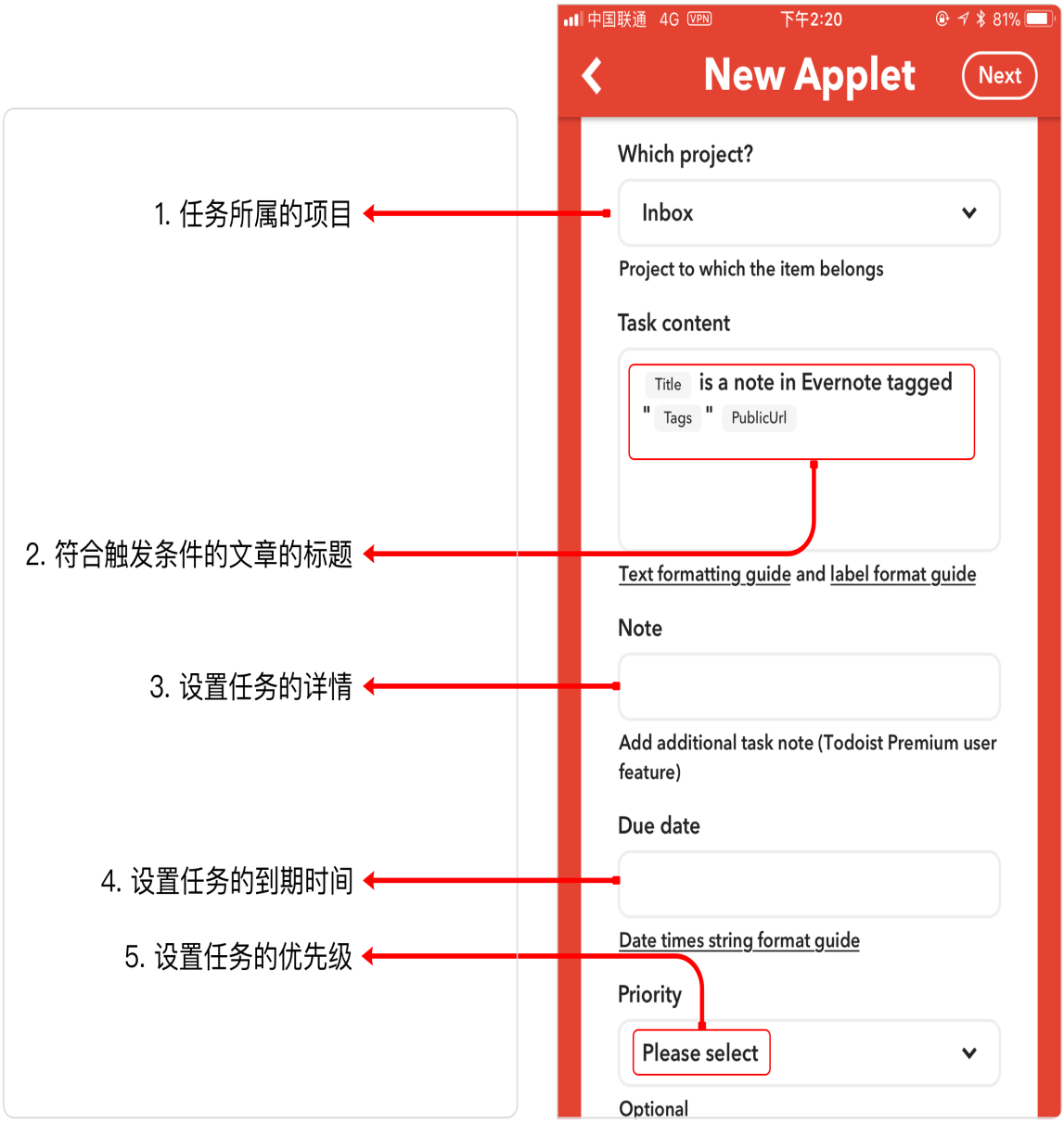
设置动作，选择服务为 Todoist，动作为「创建新的任务」



设置 Action：

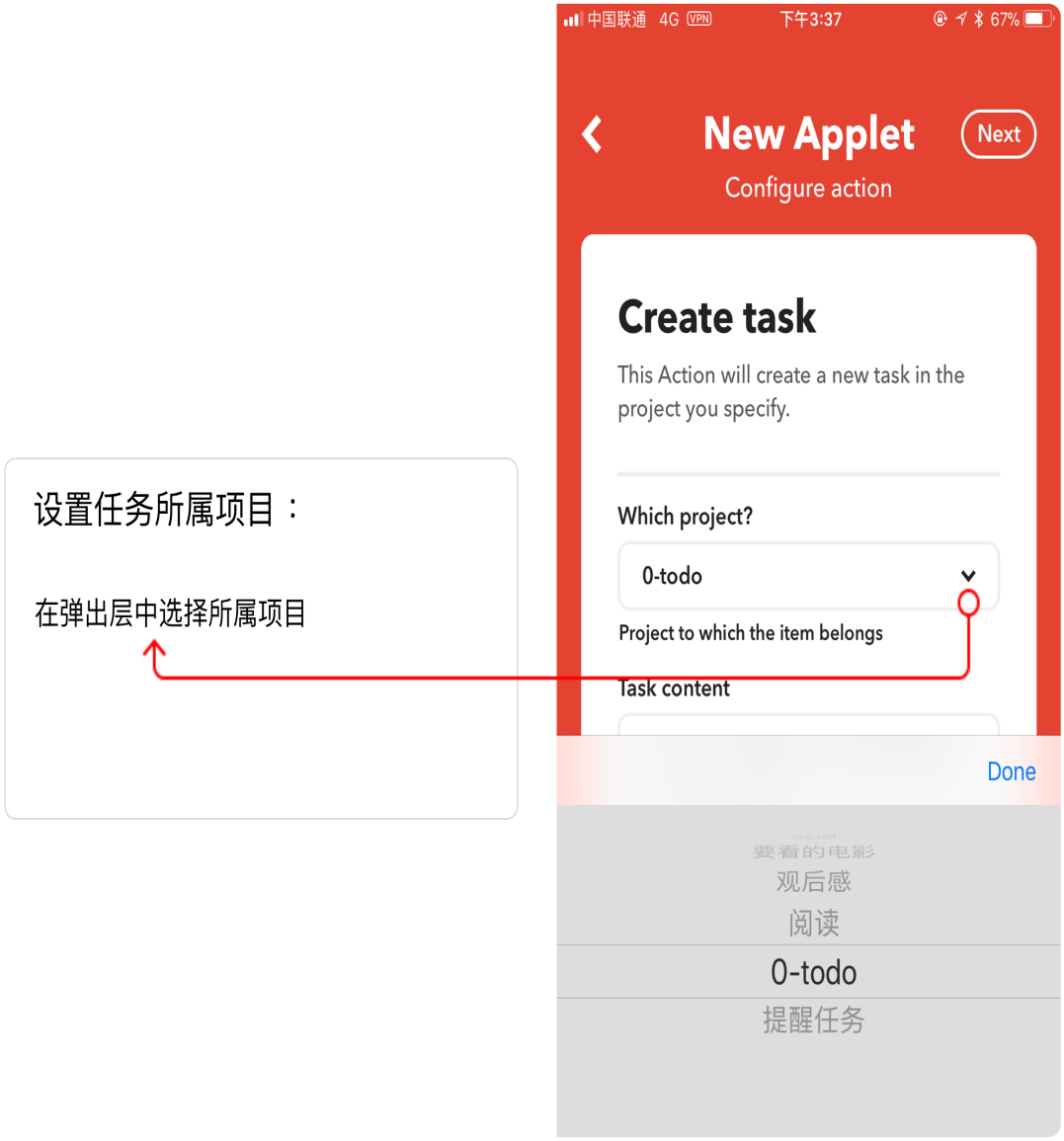
动作为「创建新任务」

IFTTT 非常智能的给我创建了默认的模版，我们先看一下内容：

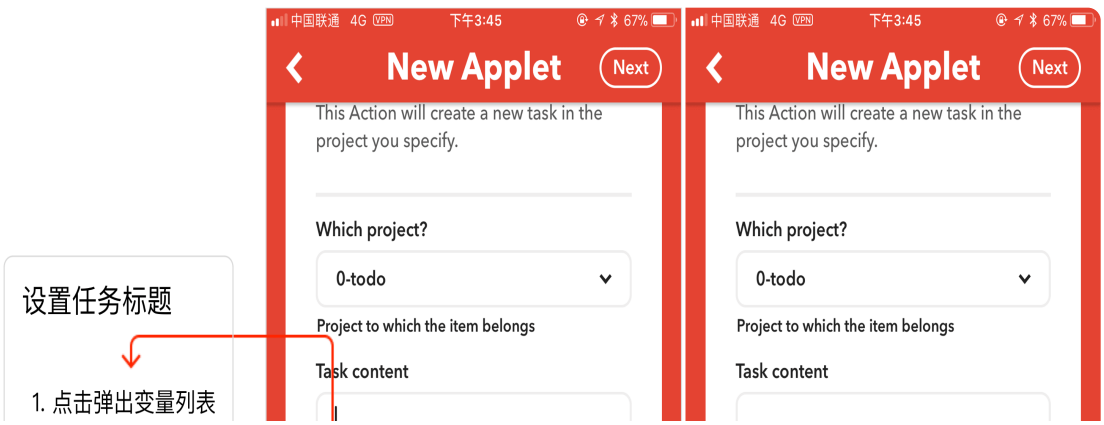


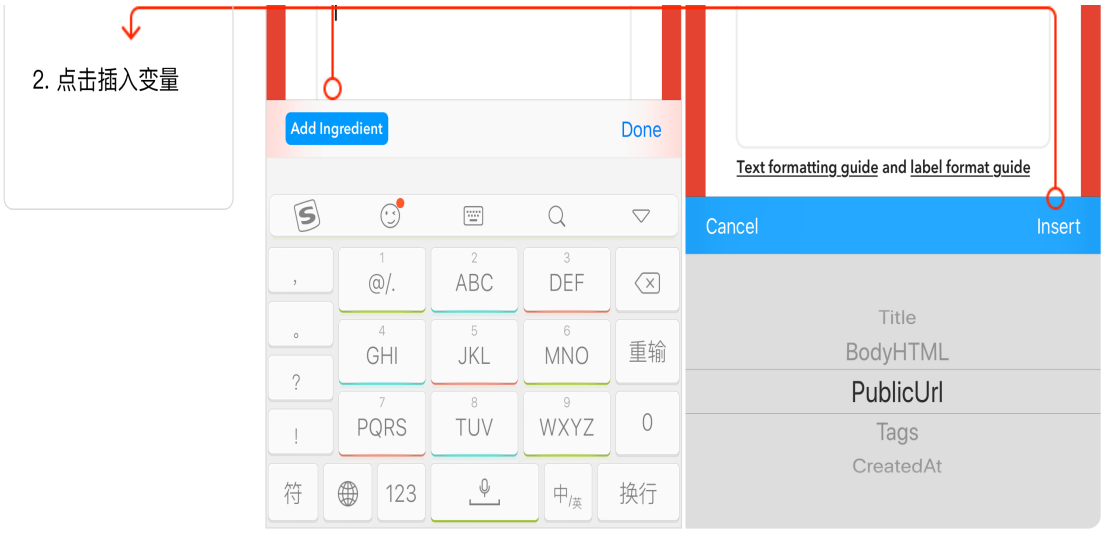
这里我们有三个要求：

1. 任务所属项目为「阅读 → 0-todo」



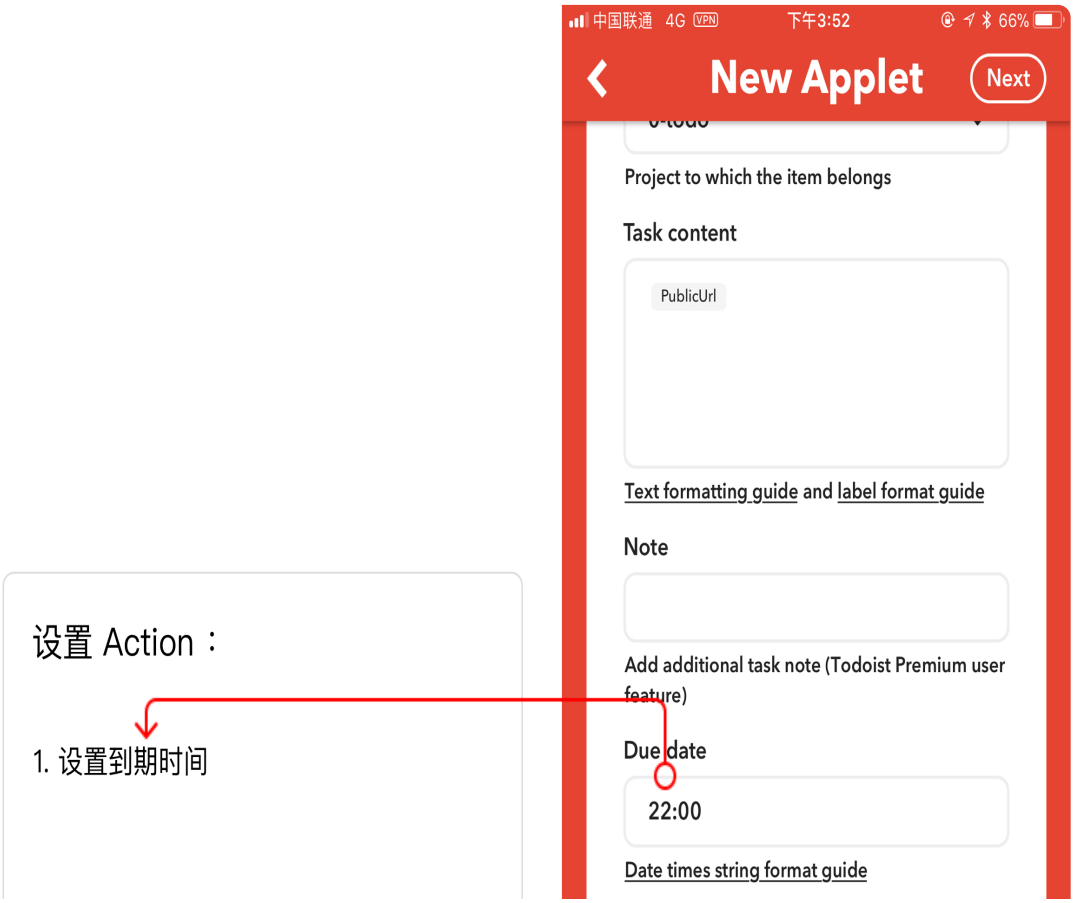
2. 任务标题为符合触发条件的文章的 URL

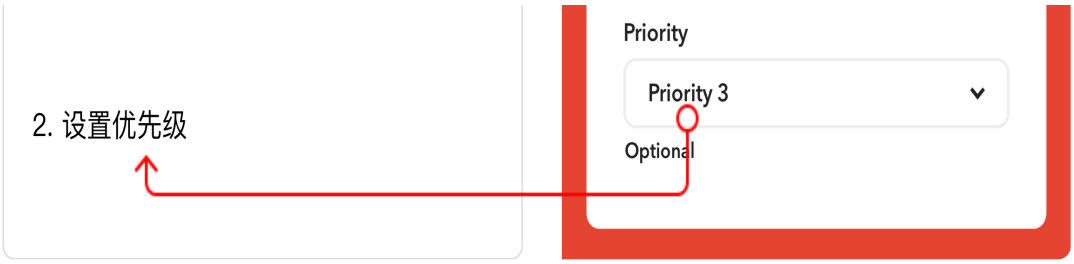




注：由于 Todoist 的标题支持 Markdown 语法，所以 **Public** 这个变量在 Todoist 中可以显示成文章标题，并且支持点击标题跳转到原文。

3. 任务提醒时间为 22:00，优先级为 3





我们看一下创建的「稍后读」任务在 Todoist 中的显示效果：

小结

我们将第二步的结果，文章保存到 Evernote 后，设置了「稍后读」标签，作为第三步的触发条件，这是一个非常巧妙的技巧，类似的如果以后我们需要创建一个复杂流程时，可以利用「每一步的触发条件是上一步的结果」这一特性，使 IFTTT 串联起来工作，极大地提升效率。

事例 1 小结

至此，完成了整个「稍后读自动化流程」的设计，我们先回顾一下，我的稍后阅读自动化流程如下：

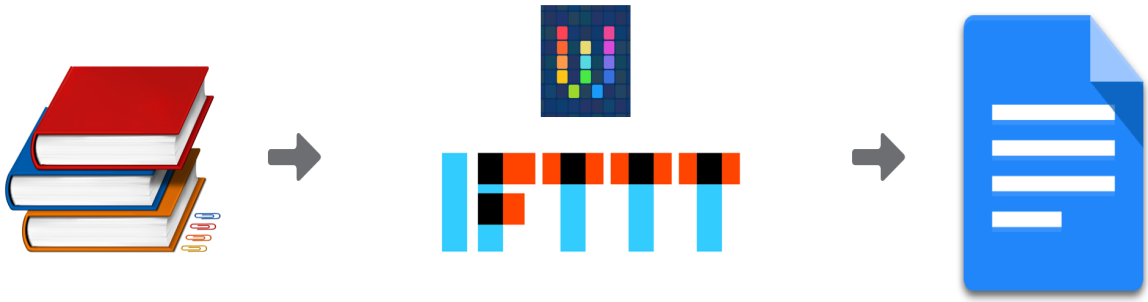
1. Feedly 遇到希望稍后读的文章，点击菜单中的收藏按钮即可。
2. 微信、Slack、Telgram、浏览器等应用中遇到希望稍后读的文章，复制文章 URL，执行 Workflow 任务即可。
3. IFTTT 发现有文章设置了「稍后读」标签后，自动去 Todoist 中创建稍后读任务。

这个流程中我最中意的就是将「稍后读」任务用 Todoist 管理起来，就像完成任务一样，读完一篇去 Todoist 中勾掉一篇，每每看到清单里的内容变少，都会觉得轻松许多。

另外，本例中，我们创建了两个自动化任务，而且第二个自动化任务的触发条件正好是第一个自动化任务的结果，这样我们相当于串联起了两个任务流，可以最大限度地发挥 IFTTT 的功效。

大家可以利用「每一步的触发条件是上一步的结果」这一特性，自由发挥创造出更加复杂的自动化任务，提高我们的生产力。

事例 2： IFTTT + Workflow 实现书籍管理自动化



目前我们接触到的 IFTTT 触发条件有两种，「云端触发」[6] 和「本地触发」[7]。上一篇文章中，IFTTT 通过 **Button**、**Note**、**Camera** 三个服务实现了本地触发，那么还有没有其他的本地触发方式呢？今天我们就介绍一种新的「本地触发」方式，通过 IFTTT 的 **Webhooks**[8]，实现 IFTTT 「本地触发」。

Webhooks，也称为反向 API，还记得 API 的概念吗？

服务 B 通过服务 A 提供的 API 接口，获取到服务 A 的数据和状态。

这里有一个非常重要的点， API 是「服务 A」定义的，然后让「服务 B」调用。而 Webhooks 正好反过来，「服务 B」可以自定义这个 API（自定义参数）。这种方式极大地拓展了服务 A 的可用范围，我们通过今天的例子来说明一下。

我的书籍管理场景：

1. 每当我买了一本书之后，将「书名」和「买书日期」写入 Google Sheets[9]。
2. 当我读完一本书，去 Google Sheets 找到上次记录，更新一下「读完时间」。
3. 最终在 Google Sheets 中维护了一个名为「我的书单」的表，里边记录了所有买过的书的「买书时间」和「读完时间」。

接下来我们利用 IFTTT 和 Workflow 实现任务自动化。

我们先看一下 IFTTT + Workflow 实现的效果：

1. 当我买了一本书，执行 workflow 脚本，输入书名，workflow 会通过 IFTTT 记录「书名」和「买书日期」到 Google Sheets 中。

2. 当我看完一本书，执行 workflow 脚本，选择书名，workflow 会通过 IFTTT 更新「读完时间」到 Google Sheets 同一条记录中。

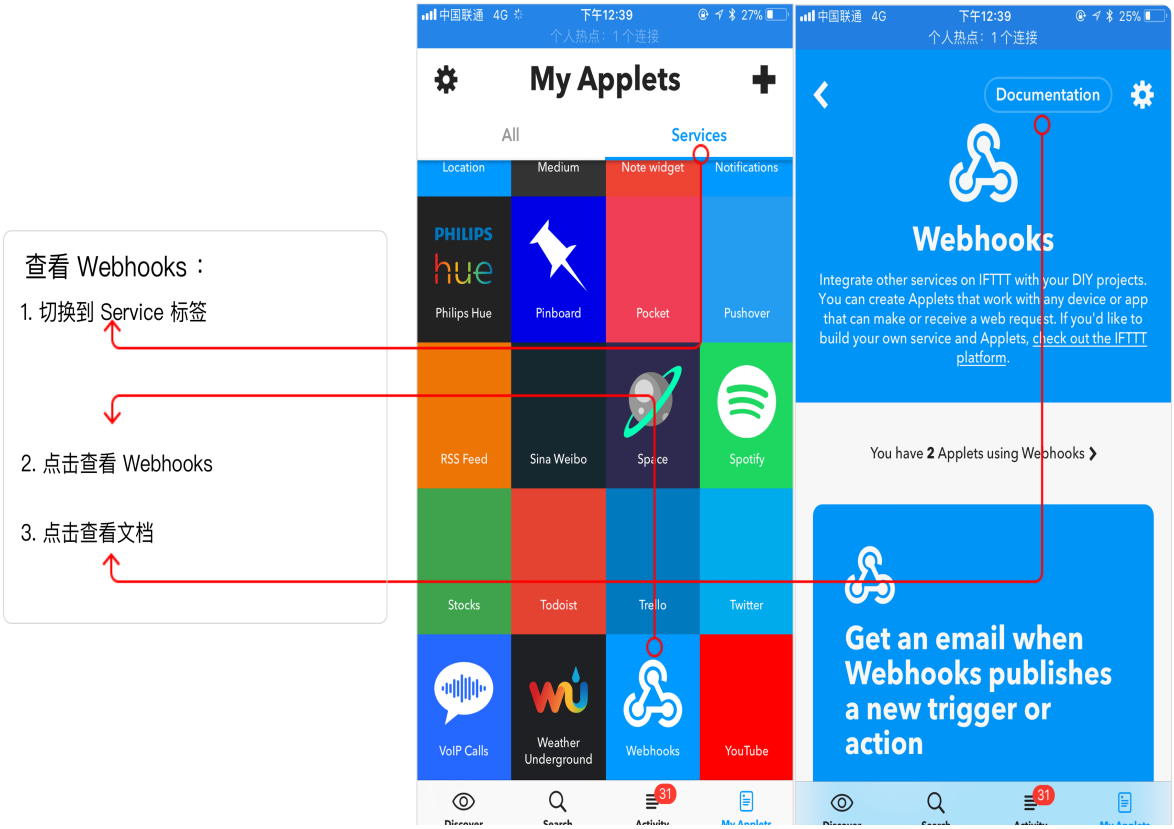
第一步，通过 Webhooks 记录「书名」和「买书日期」到 Google Sheets

我们分析一下这个需求，买书这个动作是一个人为动作，不可能在手机上有现成的 API，既然没有 API 如何通知 IFTTT 呢？这里就用到了 Webhooks。

Webhooks 相当于 IFTTT 允许我们自定义 API，对于没有 API 的触发动作，我们通过自定义 API 就可以使得 Trigger 被触发，从而执行接下来的 Action。

在 IFTTT 中每个人都有一个唯一 key，这个 key 是组成这个自定义 API 的重要部分，IFTTT 通过这个 key，区分一个 Webhooks 是哪个用户创建的，大家可以点击链接查看自己的 key。

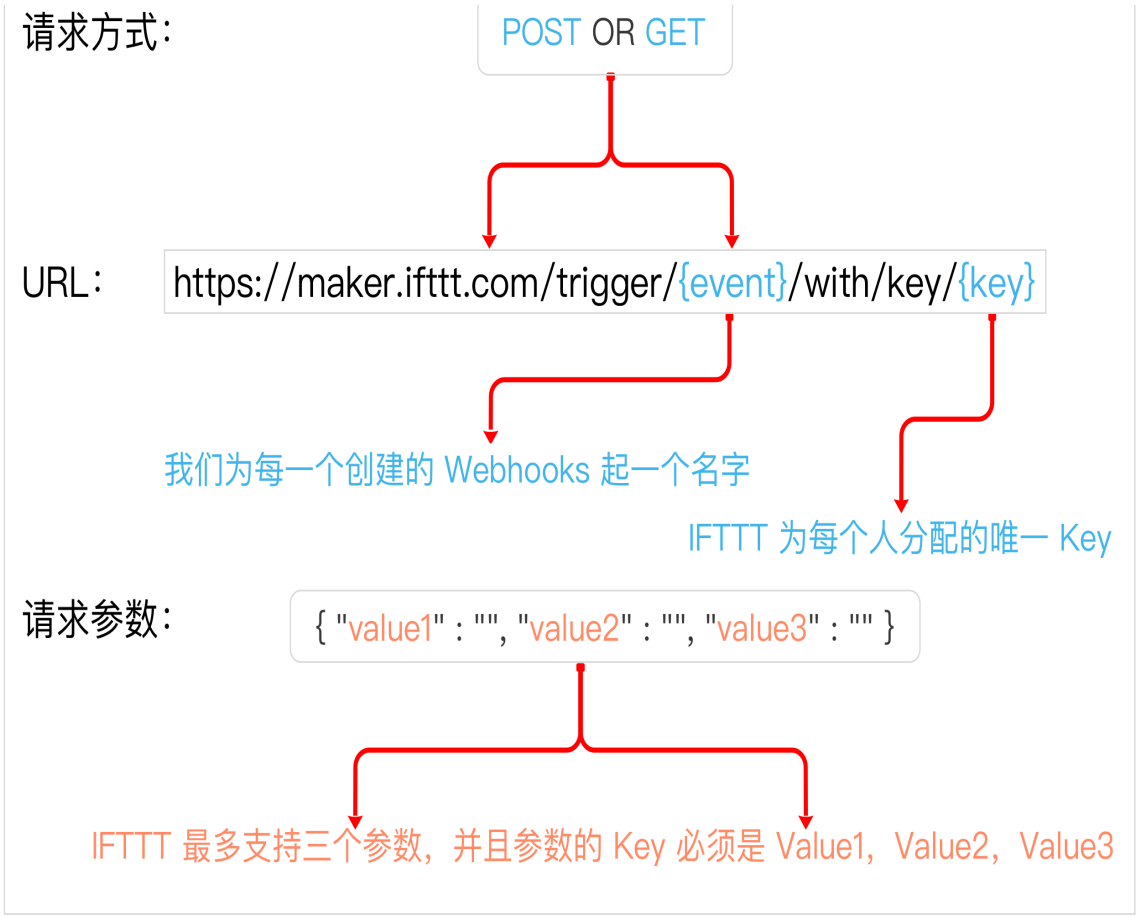
也可以如图通过 IFTTT 应用查看自己的 key。



请大家记住这个唯一 key 值，接下来我们会用到。

Webhooks 使用原理

IFTTT 为每个 Webhooks 提供了一段唯一的地址，并且对 Webhooks 的请求方式，请求参数做了严格的定义。



如果懂 HTTP 的同学应该很清楚怎样使用 Webhooks 了吧，其实很简单，我们只需要通过 POST 或 GET 方式请求 IFTTT 分配的 URL，然后最多带三个参数就能够使用 Webhooks 了。

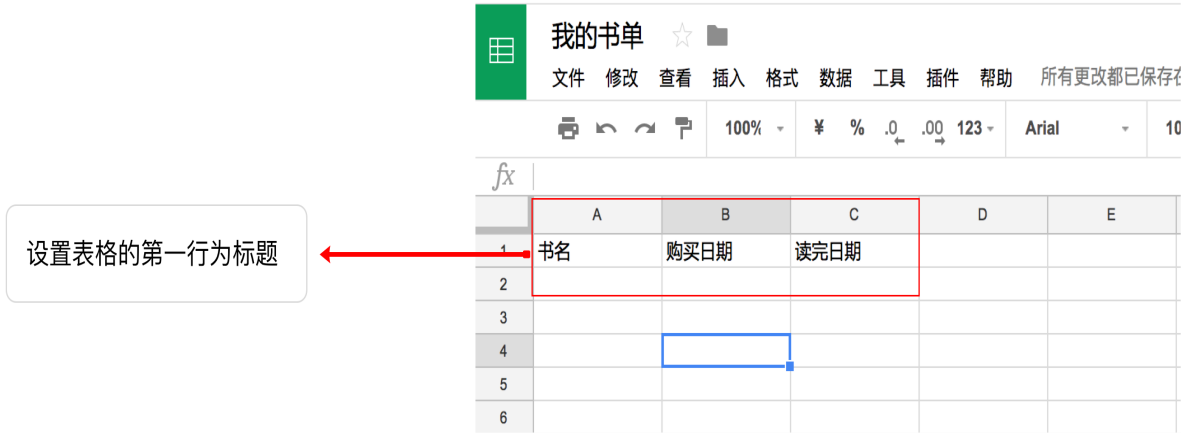
有了 Webhooks 我们就相当于有一个 **Trigger** 了，只要请求 Webhooks 的地址，IFTTT 就会被触发，接下来就可以自动执行设置好的 **Action** 了。

那么作为普通用户（非开发人员）如何发起 POST 或 GET 请求呢，那就要再次使用 iOS 平台上的神器 Workflow。通过 Workflow 我们可以在 iOS 客户端发起请求，让 IFTTT 自动工作。

接下来我们以「通过 Workflow + IFTTT 记录买书时间到 Google Sheets」为例看看如何使用 Webhooks 创建自动化任务。

创建初始表

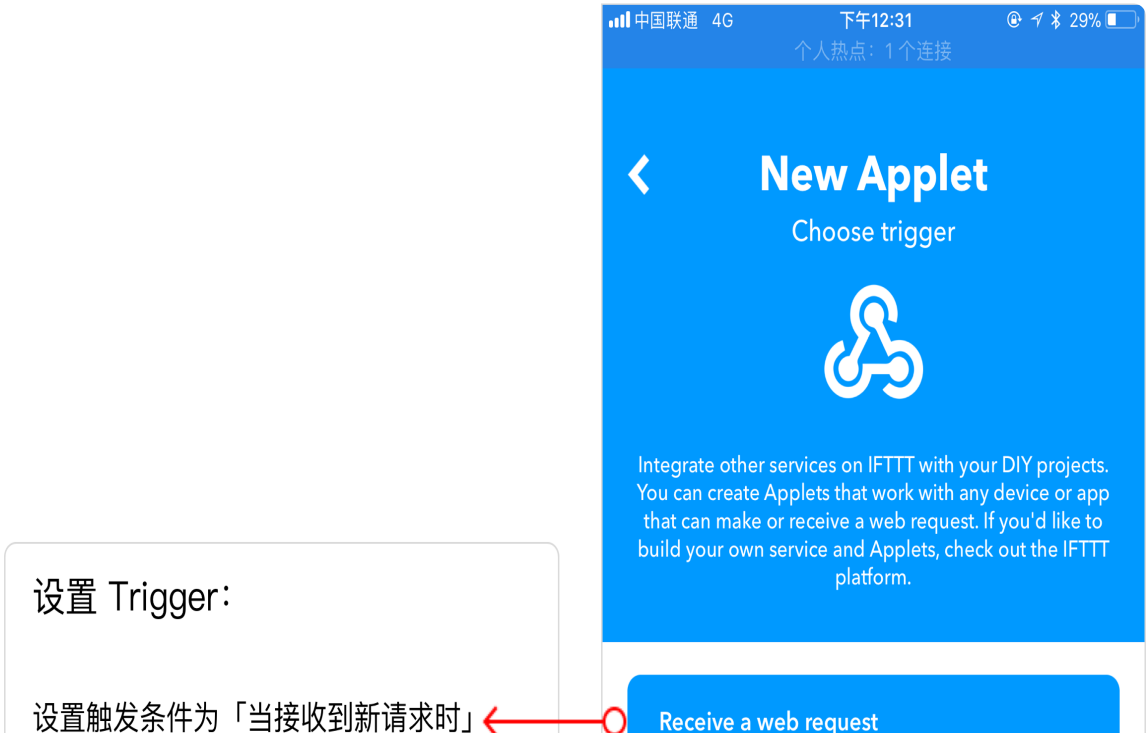
在创建这个自动化任务之前，我们先去 Google Sheets 的 `\IFTTT` 路径下创建一个名叫「我的书单」的表，用来记录我们的书籍信息。



接下来我们开始创建这个自动化任务。

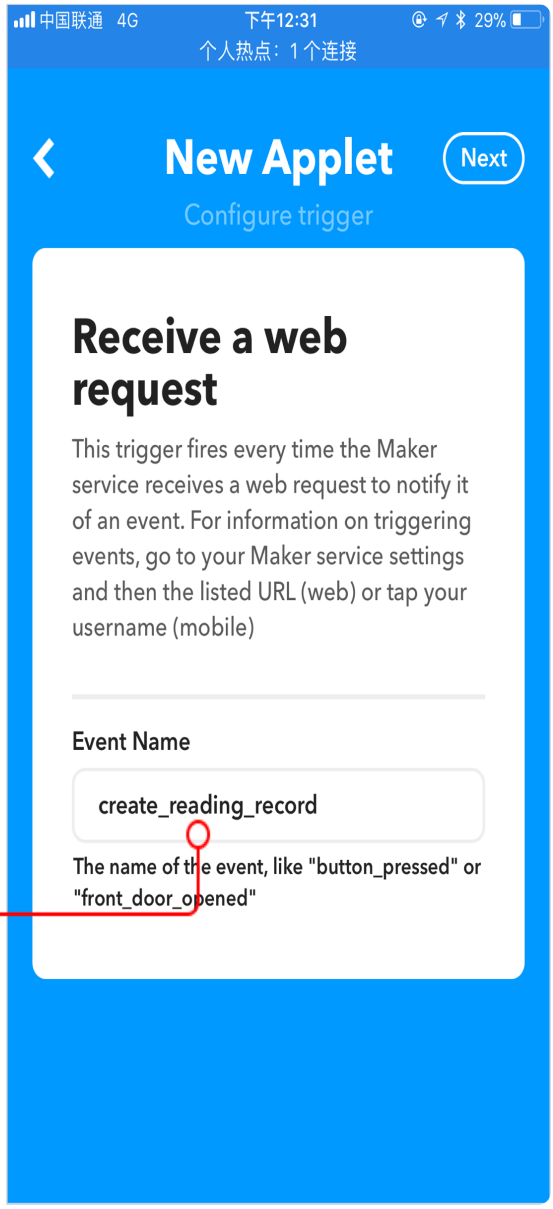
设置 Trigger

设置触发器，选择服务为 Webhooks，设置条件是「当接收到新的请求时」。



设置 Trigger：

1. 设置 Webhooks 的 event


中国联通 4G
下午12:31
29%
个人热点：1 个连接
New Applet
Next
Configure trigger
Receive a web request
This trigger fires every time the Maker service receives a web request to notify it of an event. For information on triggering events, go to your Maker service settings and then the listed URL (web) or tap your username (mobile)
Event Name
create_reading_record
The name of the event, like "button_pressed" or "front_door_opened"

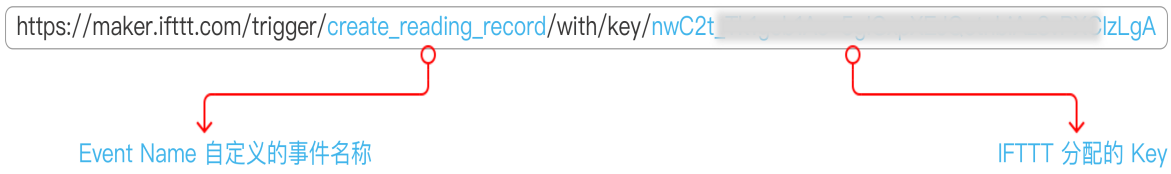
我们注意到上图中需要设置 Webhooks 的 **Event Name**，这就是上文中我们提到的 URL 中的 `{event}`，这里我们设置 `{event}` 值为 `create\_reading\_record`。

为什么 **Event Name** 的命名是这样的呢?

由于 Webhooks 实际上是定义了一个 URL，而 **Event Name** 又是 URL 的一部分，所以必须满足 URL 的定义：

1. URL 请求采用小写字母，数字，部分特殊符号（非制表符）组成。
2. URL 请求中不采用大小写混合的驼峰命名[10] 方式，尽量采用全小写单词，如果需要连接多个单词，则采用连接符 `_` 连接单词。

这一步中，我们创建了 Trigger，选择服务为 Webhooks，并设置 Event 为 `create_reading_record`。这样 IFTTT 就为我们分配了一个唯一的 Webhooks 地址：



这也意味着我们只要对上边 URL 发起 **POST** 或 **GET** 请求，触发器就会触发。

设置 Action

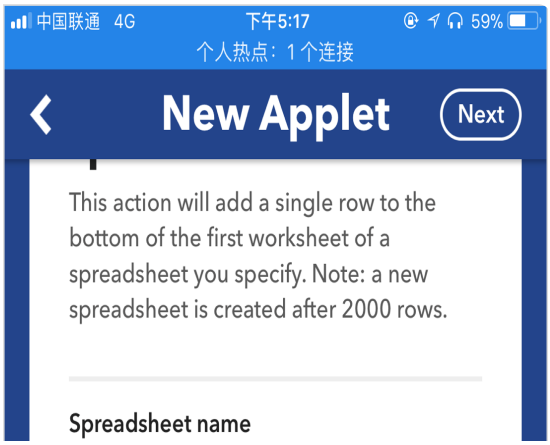
先明确一下我们要做的事情，我们需要当触发器被触发的时候，自动在 Google Sheets `\IFTTT` 目录下的「我的书单」表中添加一条记录，记录「书名」和「买书时间」。

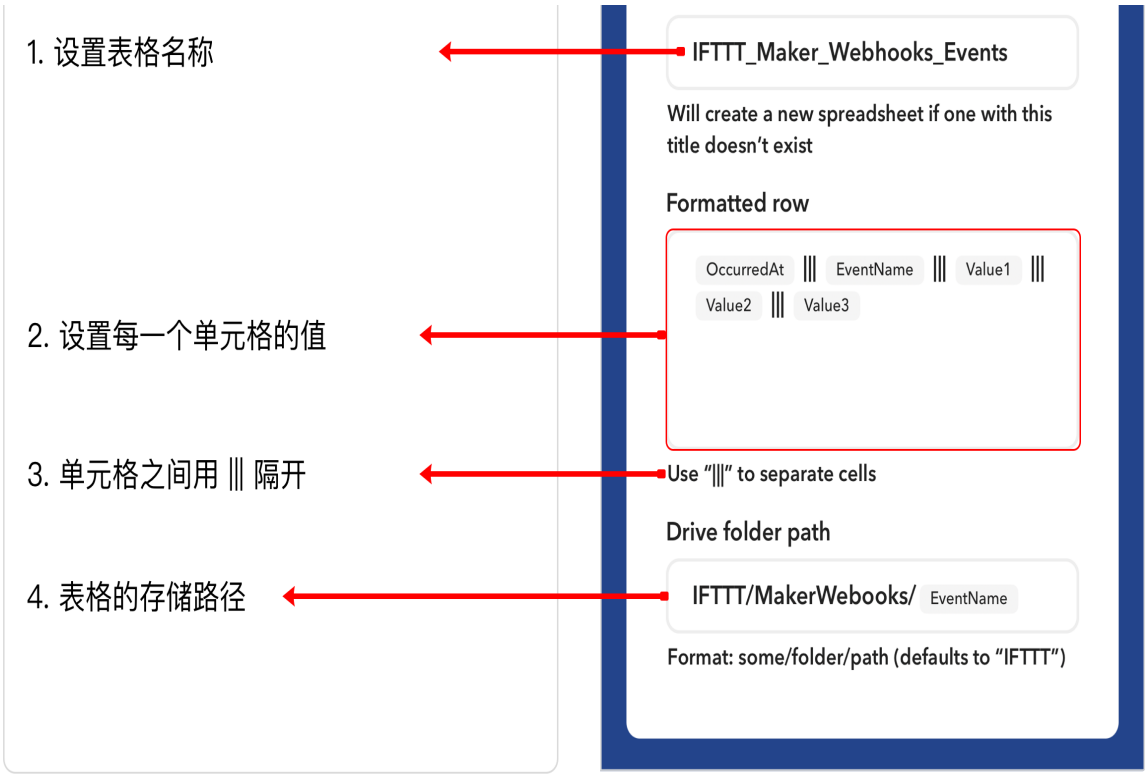
设置 Action：

设置动作为「添加一行新数据」

我们看一下 IFTTT 提供的原始动作模板：

设置 Action：



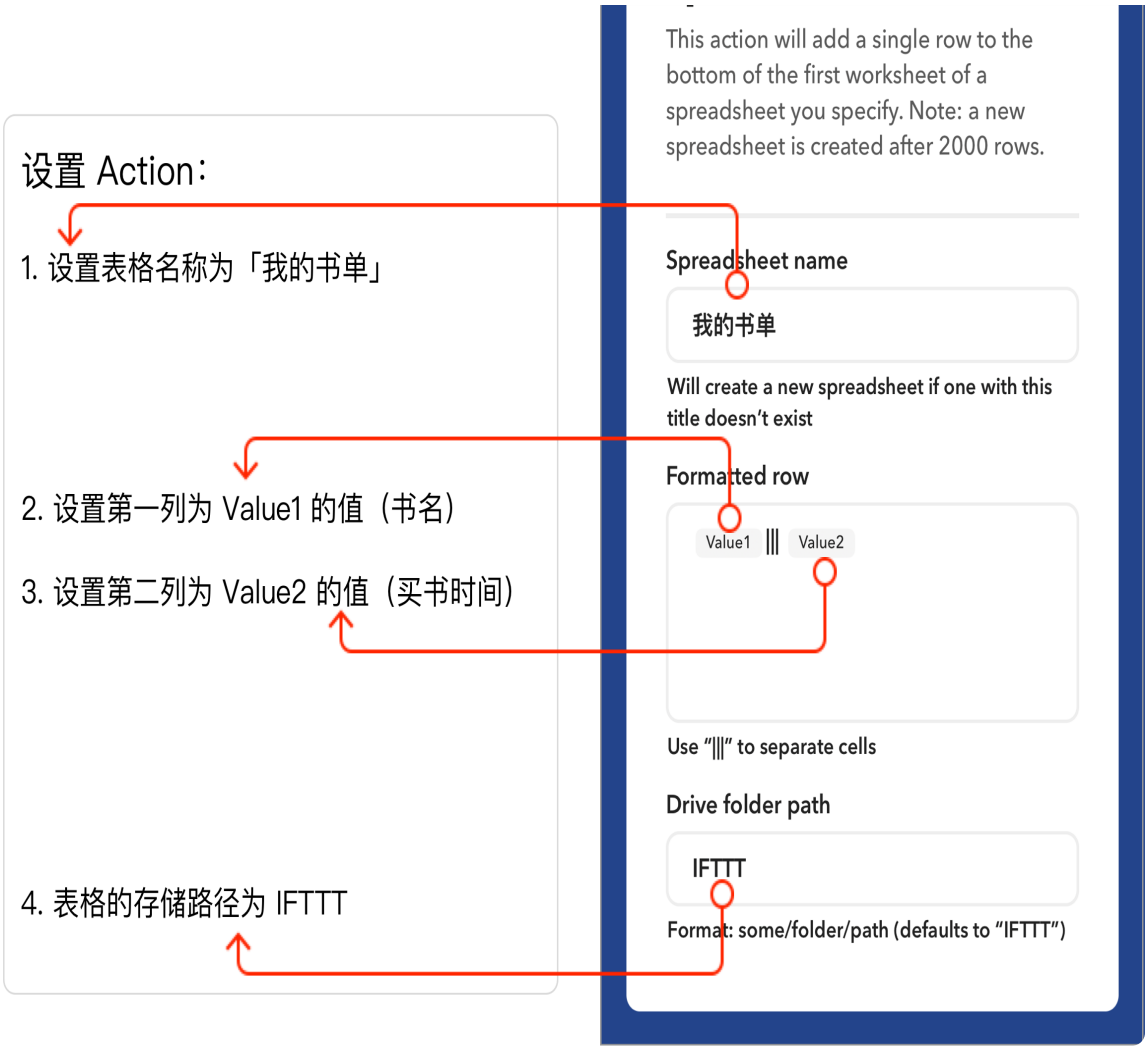


上图中 IFTTT 默认 Webhooks 的相关参数写入到表格模板里，包括请求时间，事件名称，`Value1-3` 的值。并且用 `|||` 区分每个单元格。

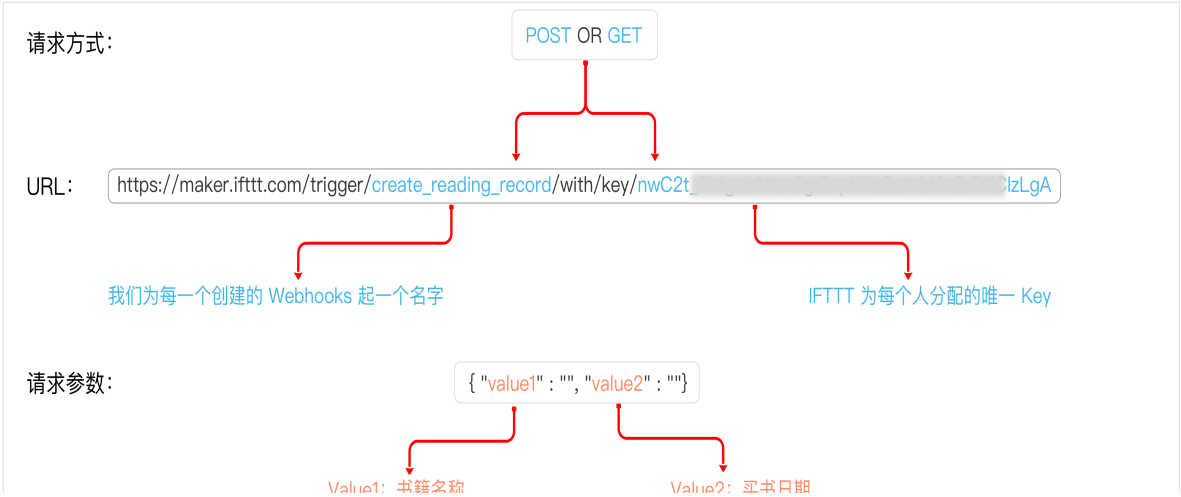
根据我们的需求，只需要做如下设置：

- **Spreadsheet name**：`我的书单`
- **Formatted row**：`Value1 ||| Value2`
- **Drive folder path**：`IFTTT`

这样设置的意思是说，如果上一步定义的 Webhooks 被请求，那么我们将请求参数名为 `Value1` 对应的值存入第一列，将请求参数名为 `Value2` 的值存入第二列。



通过如上设置，如果我们请求 Webhooks，IFTTT 就会被触发，并且将「请求参数」中的 **Value1（书名）**和 **Value2（买书日期）**，写入 Google Sheets。

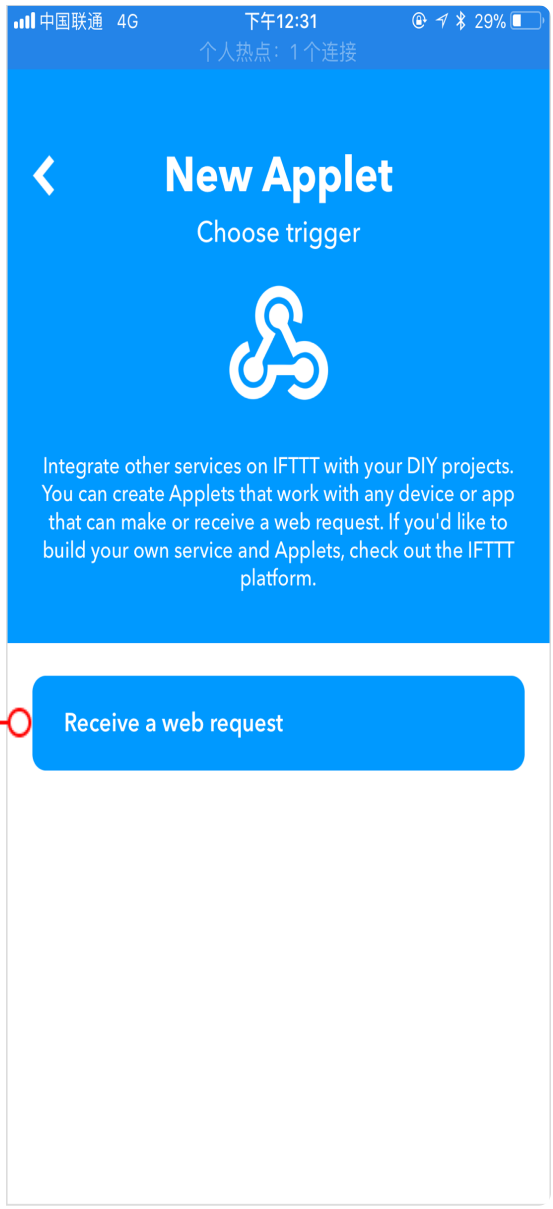


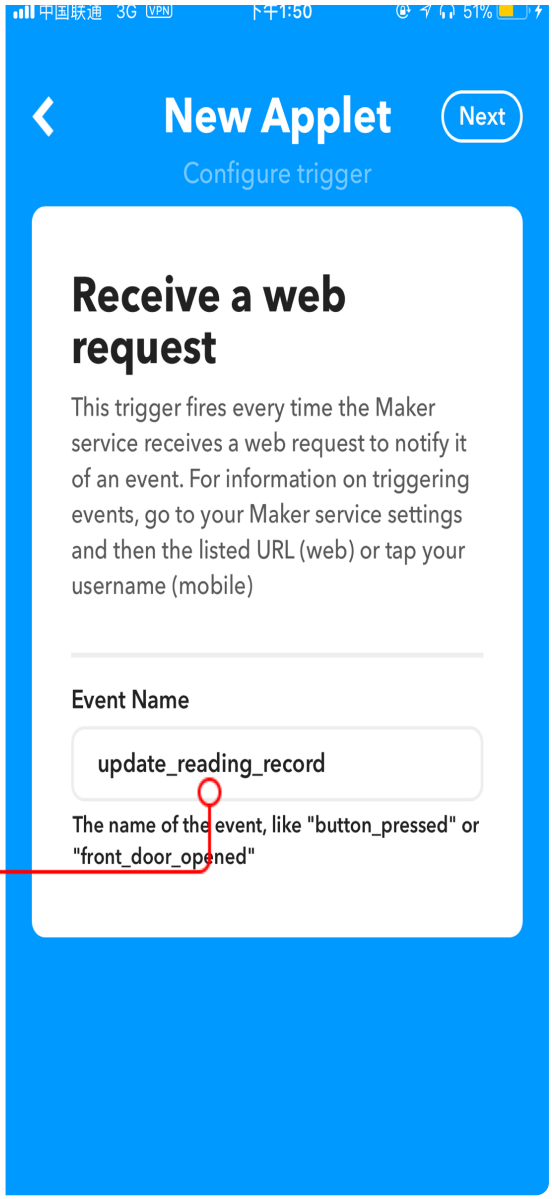
第二步，Workflow 请求 Webhooks 更新「读完日期」

我们需要在 IFTTT 创建一个 Webhooks，用来更新书籍的「读完日期」。

设置 Trigger

设置触发器，选择服务为 Webhooks，设置条件是「当接收到新的请求时」。

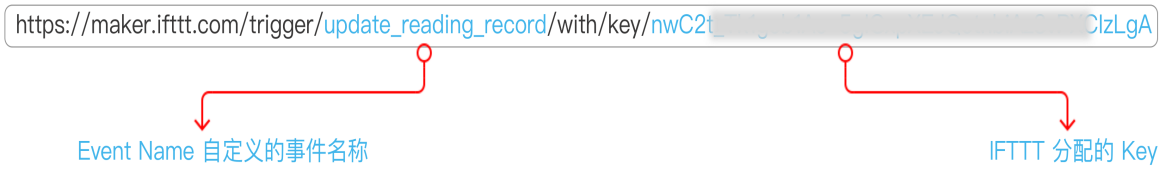



New Applet
Next
Configure trigger
Receive a web request
This trigger fires every time the Maker service receives a web request to notify it of an event. For information on triggering events, go to your Maker service settings and then the listed URL (web) or tap your username (mobile)
Event Name
update_reading_record
The name of the event, like "button_pressed" or "front_door_opened"

设置 Trigger：

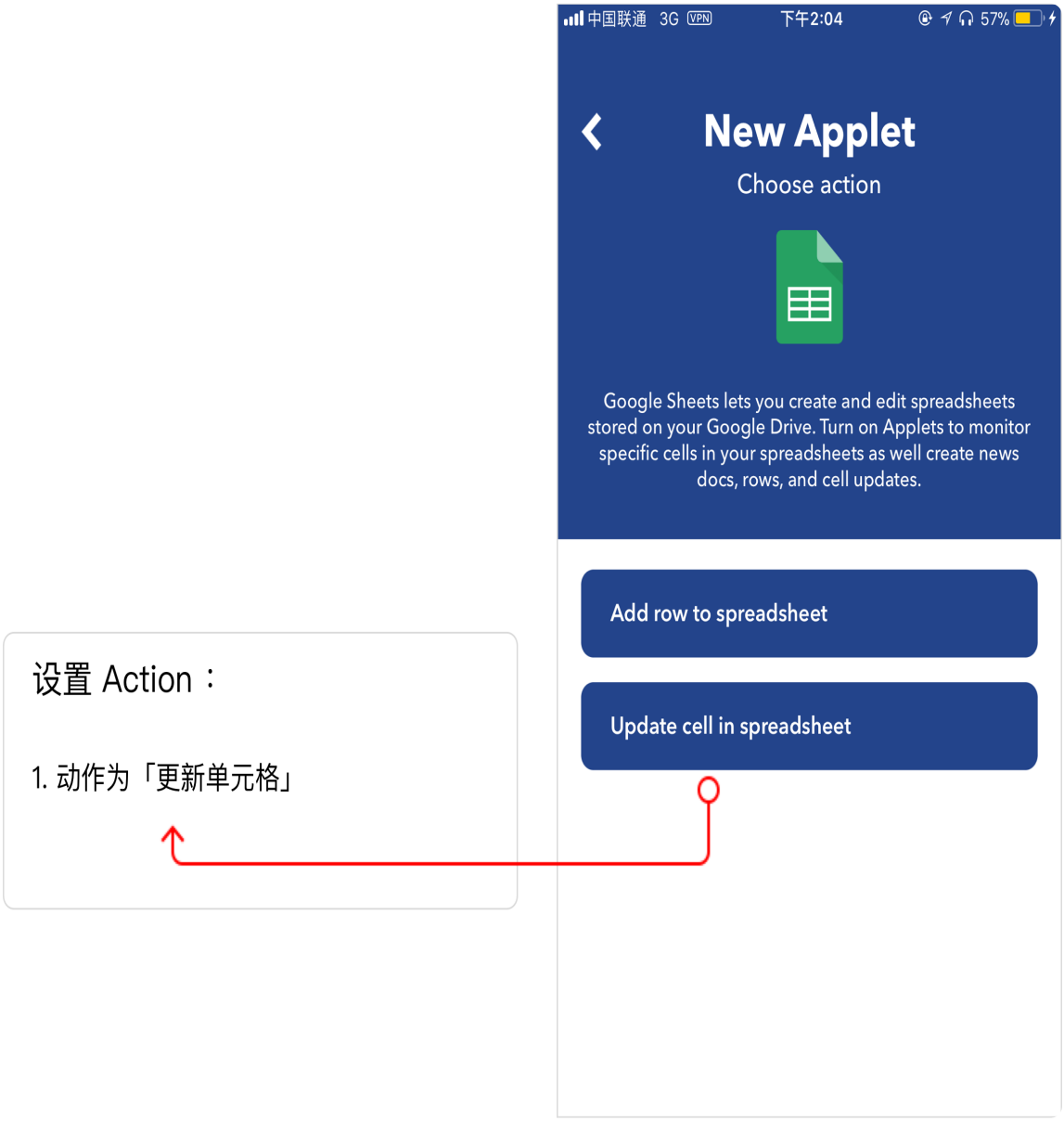
1. 设置 Webhooks 的 event

我们设置 **Event Name** 为 `update\_reading\_record`，所以我们的 Webhooks 地址如下：



设置 Action

我们需要设置当触发器被触发的时候，去 Google Sheets 中「更新」一行数据，记录「读完日期」。



接下来我们将设置 Action「模板」，在设置之前先跟大家讲一下 IFTTT 是如何更新 Google Sheets 信息的。

简单说就是：**IFTTT 是通过单元格的「坐标值」更新 Google Sheets 信息。**

那什么是坐标值呢?

我们知道在表格中，一个单元格的「列号」是以英文字母命名的，单元格的「行号」是以数字命名的，比如我们想更新第一列第一行的单元格，那么他的坐标就是 `A1`，同理第三列第二行单元格是 `C2`，看下图：

既然 IFTTT 需要「坐标值」才能更新 Google Sheets，我们就必须在更新书籍信息的时候知道「这本书」对应在 Google Sheets 中是哪行记录，从而得出他的「坐标值」。

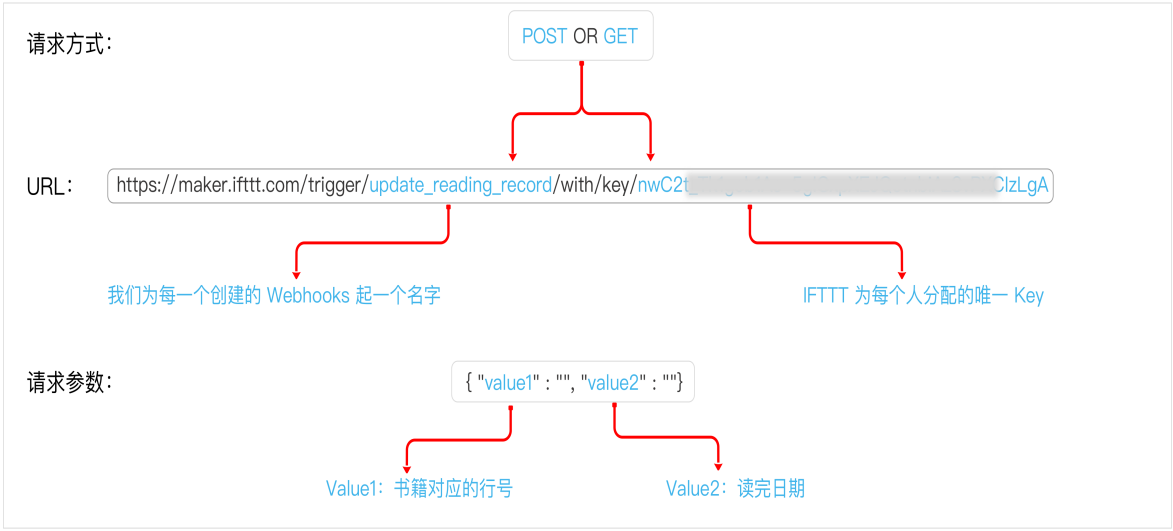
还是以上图为例，我们想更新「我的第二本书」的「读完日期」，可以看到「读完日期」的列号是 `C`，接下来我们要想办法知道「我的第二本书」的「行号」，就可以通过 IFTTT 更新了，很明显「我的第二本书」的「行号」是 3，所以「我的第二本书」的「坐标值」就是 `C3`。

对于任意一本书，如果我们要更新它的「读完日期」，根据上图我们可以明确

对于任意一本书，如果我们要更新它的「读完日期」，根据上图我们可以明确得出他的列号一定是 `C`，那么再知道他的「行号」我们就可以定位到「坐标值」了。

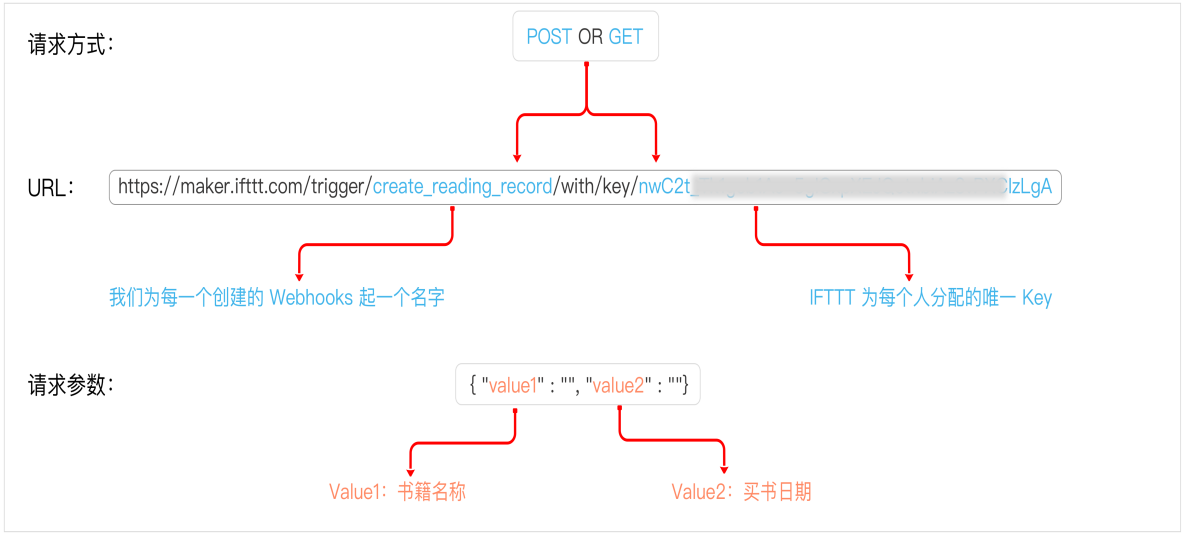
这个「行号」就需要我们通过参数传递给 Webhooks 了，所以我们定义第一个参数也就是 `Value1` 的值为行号，即待更新的「坐标值」为 `C{Vlaue1}`，如图：

通过如上设置，我们只要请求 Webhooks，IFTTT 就会被触发，将请求参数中的 **Value1（书的行号）**和 **Value2（读完日期）** 写入到 Google Sheets 中。



第三步，Workflow 请求 Webhooks 记录「书名」和「购买日期」以及「读完日期」

在第一步中我们创建了一个 Webhooks 记录「书名」和「购买日期」：



而且我们已经明确了需要提交两个参数：

```
Value1 =「书名」和 Value2 =「买书日期」
```

剩下的事很简单了，每当我们买了一本书后，通过 Workflow 在 iOS 上请求 Webhooks，传递「书名」和「买书日期」两个参数即可。

为什么用 Workflow 有两个原因:

1. 我需要一个随手就能请求 Webhooks 的工具，方便我买书以后快速记录，在 iOS 系统上 Workflow 效率非常高。
2. 由于「IFTTT 使用 Google Sheets 的限制」，我需要 Workflow 调用 Webhooks 的同时再存储一条数据到 Evernote （这条接下来解释）。

IFTTT 使用 Google Sheets 的限制

我们先回顾一下需求：

1. 每当我买了一本书之后，将「书名」和「买书日期」写入 Google Sheets 。
2. 当我读完这本书，去 Google Sheets 找到上次记录，更新一下「读完时间」。

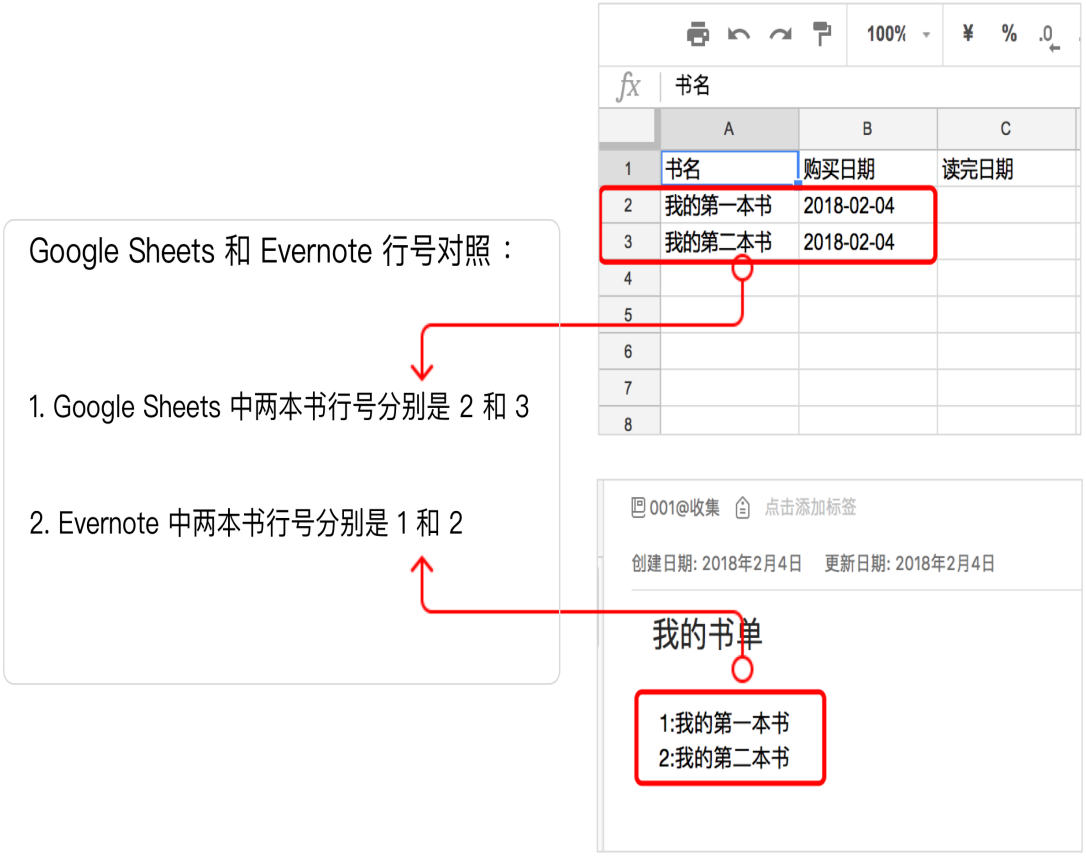
第一步比较简单，我们通过 IFTTT 调用 Webhooks， IFTTT 会自动记录「书名」和「买书日期」到 Google Sheets。但是第二步就比较麻烦了，这里涉及到更新问题，上边我们提到过 IFTTT 是通过「坐标值」更新 Google Sheets 记录的。

可是在 IFTTT 中并没有提供一个方法让我们通过书名就能得到它在 Google Sheets 中是第几行第几列也就是所谓的「坐标值」，不知道「坐标值」怎么更新记录呢?

这里我用了一个讨巧的办法：

1. 每次通过 Webhooks 记录「书名」和「买书日期」到 Google Sheets 后，

同时记录到 **Evernote** 的「我的书单」笔记中。这样之后，同一本书在 Google Sheets 和 Evernote 中「行号」正好相差 `1`，如下图：



2. 以「我的第一本书」为例，当读完这本书后，可以通过 Workflow 获取 **Evernote** 中「我的书单」笔记内容，然后利用 Workflow 过滤找到「我的第一本书」这一行记录，得到这本书的序号是 `1`。根据「Google Sheets 和 Evernote 中同一本书的行号相差 `1`」这个公式，我们计算得出「我的第一本书」在 Google Sheets 的行号是 `1+1 =2`。
3. 有了「行号」就可以通过 IFTTT 的 Webhooks 更新 Google Sheets 的记录了。

由于本系列主要介绍 IFTTT 用法，所以关于 Workflow 用法这里不做展开，大家可以通过以下链接获取 Workflow 脚本自行研究：

- IFTTT 记录购书日期
- IFTTT 记录读完一本书日期

事例 2 小结

本例中，我们第一次尝试使用了 IFTTT 的 Webhooks ，自定义了一套规则，实现了调用 Webhooks 将「购书日期」和「读完日期」记录到 Google Sheets 的需求。

Webhooks 是一个非常强大的功能，它意味着我们可以定制 API，让 IFTTT 更加智能地为我们服务。想一想如果你有一个独立可控制的「服务器」，我们就可以制定各种 Webhooks ，让 IFTTT 我们自动化处理任务。比如智能家居控制，比如服务器报警，比如远程下载，比如远程监控等。在后续的文章中，我们还将多次使用 Webhooks，大家敬请期待。

本篇总结

本篇是玩转 IFTTT 系列教程的第四篇**进阶篇**，文章中我们尝试了多个 IFTTT 任务串联完成一个复杂的流程，也第一次引入了 Webhooks 的概念。掌握一个内容最基本的流程是消化 → 吸收 → 输出，希望大家多多尝试创建自己的自动化任务。

在前几期中，我们由浅入深的学习了 IFTTT 的基本用法和高阶用法，下一篇文章中，我们将智能硬件和 IFTTT 结合起来，带领大家体验一些更 Geek 的玩法，大家敬请期待。

---

1. Feedly，优秀的 RSS 服务提供商。 ↩
2. Todoist，一款优秀的 GTD 软件，丰富的 API 具备很强的扩展性，同时支持语义化创建任务。 ↩
3. 触发器，指用户设置的触发条件，IFTTT 会以轮询的方式监控用户设置的触发条件是

否达成。 ↩

4. 动作，指用户设置好的一系列动作内容，当 IFTTT 发现触发条件达成，就会继续执行用户设置好的动作。 ↩

5. Share Sheet，iOS 系统级的分享功能，任何应用都可以接入，一旦接入，无需特殊编码就可以将文字、图片或者链接分享到其他应用。 ↩

6. 云端触发，在「如果 A 条件被满足，那么就让 B 事件发生」的事例里， IFTTT 通过云端轮询的方式调用 A 条件的 API，判断 A 条件是否满足，然后才执行 B 操作，这种触发方式我们称为「云端触发」。 ↩

7. 本地触发，在「如果 A 条件被满足，那么就让 B 事件发生」的事例里， IFTTT 无法在云端监控到 A 条件被满足，需要借助 IFTTT 应用，才能发现 A 条件被满足，然后执行 B 操作，这种触发方式我们称为「本地触发」。 ↩

8. Webhook，准确的说 Webhook 是一种 web 回调或者 http 的 push API，是向 APP 或者其他应用提供实时信息的一种方式。Webhook 在数据产生时立即发送数据，也就是你能实时收到数据。这一种不同于典型的 API，需要用实时性需要足够快的轮询。这无论是对生产还是对消费者都是高效的，唯一的缺点是初始建立困难。 ↩

9. Google Sheets，Google 出品的云端数据表，提供了非常优秀的 API，方便我们在云端处理、统计和计算。 ↩

10. 驼峰命名法，是电脑程式编写时的一套命名规则（惯例）。通常约定如果出现多个单词混合命名时，首词开头字母小写，后边每个单词开头字母均大写。如，myHome，redingRecord 等。 ↩

11. 这里注意 Webhooks 的定义规范，参数名必须是 Value1，Value2 和 Value3。 ↩